Static Typing in Python With MyPy

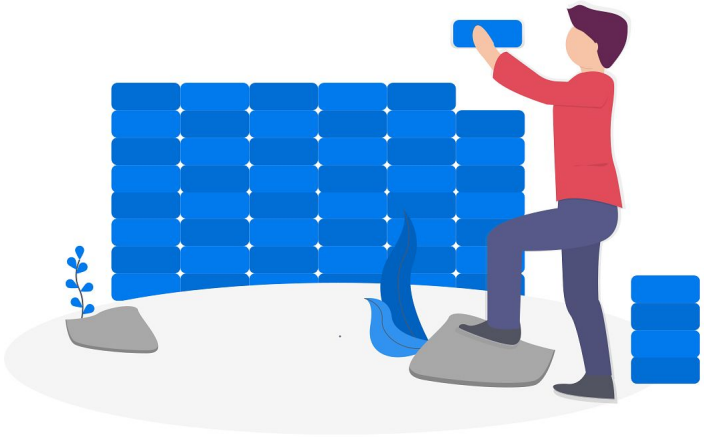
Amitosh Swain Mahapatra
Product Engineer, Gojek Tech

$ whoami

- Product Engineer @ Gojek
- Contributes to Open Source, makes cool things
- Sometimes draws
- Find my code on Github @ agathver
- Catch me on Twitter @agathver
- Lives in Bangalore, India

Agenda

- What is static typing
- Features of Python static typing
- Duck Typing
- How to use typings in your project



What is Static Typing

Let's get the boring stuff out of the way before we go ahead.

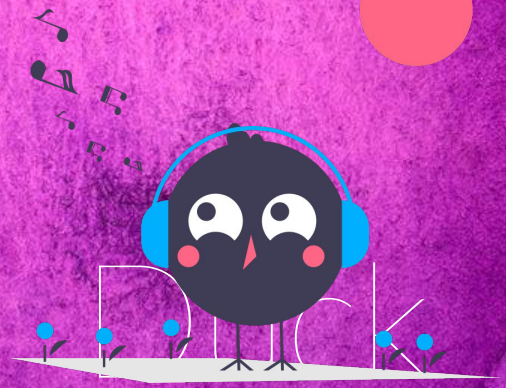
What is Static Typing

- You know what are the parameters and what is being returned by a method
- What kind of object does a variable hold
- What are the methods and properties does an object of a given type have
- All these information is known at compile-time

Example: len() always takes a Sequence as input and returns an integer

Static Typing in Python

- Python is dynamically typed, which is powerful but frequently error prone.
- You can pass anything anywhere – “duck typing”
- Programmers felt the need to specify a set of expectations for a method return value or argument
- PEP-484 was born.
- Allowed optional type-hints (based on annotations)
- No runtime type checking, Python remains weakly typed in its core, unlike Java



Typing

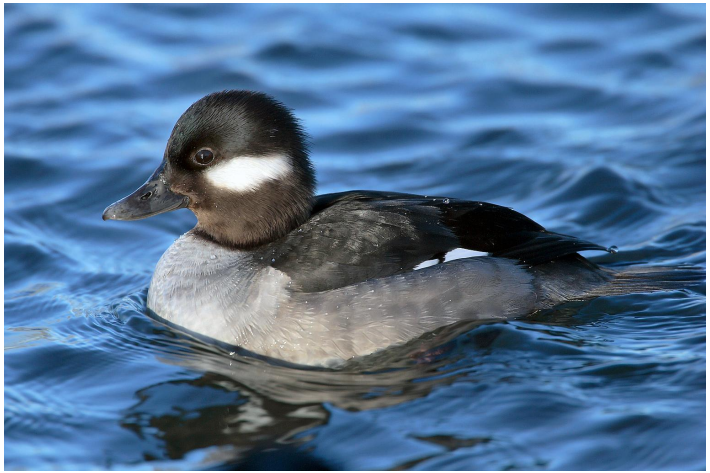
If it walks like a duck and quacks like a duck, it's a duck

Duck Typing

This is a bufflehead duck (below)

- It quacks
- Has webbed feet

Hence, it's a duck



This is a mallard (above)

- It quacks
- Has webbed feet

Hence, it's a duck

The Story of the Duck Simulator

- You and several teams are building a multi-billion dollar duck simulator to predict how ducks will change the world
- You are not just experimenting with duck, but several duck-like birds, like mallards
- You have multiple teams, one building the simulation and others simulating the attributes of ducks, mallards etc.
- Since ducks and mallards are quite similar, your simulator just treats everything as "*Ducks*" and it works. For now.

Teams are busy building ducks...

- You decided to implement `fly()` in the simulator, mailed the change in spec to every team
- Every team implemented the change except the team handling the Fuegian Steamer, because the ducks, well they don't fly.
- Their tests passed (as they test with mocks) and you decided to run the simulation with all the ducks combined.
- Simulation crashed, your lab caught fire, Cthulhu arrived

With big teams and big code bases, mistakes like this may happen

Your Brain can only store so much information. So why not automate this.

MyPy

MyPy is a static type-checker for Python using PEP-484 type annotations. It runs as part of your build / lint process and has 0 runtime overhead

How to use types in Python

```python
from typing import Sequence

def do_magic(iterable: Sequence) -> int:
    # ... some magic happens here
    return 42
```

Describes a method that takes a `Sequence`, a type that implements `__iter__` and returns an `integer`

```python
@dataclass
class InventoryItem:
    """Class for keeping track of an item in inventory."""
    name: str
    unit_price: float
    quantity_on_hand: int = 0

    def total_cost(self) -> float:
        return self.unit_price * self.quantity_on_hand
```

Defines a class with properties name, unit_price and quantity_on_hand as a string, float and int respectively.

You can also describe variables, but that's generally unnecessary

```python
variable: str = "Hello"
```

The Python type system

Classes

Any Python class *like int, str, Path, Logger etc.* can be used as a type for a method, field, return or a variable

Compile-type only with runtime type information

Python is still, at its core a dynamic typed language.

Generics

Python types support Generics. Few inbuilt classes have been generified using classes from typings like List[str], Dict[str, int]

Inheritance-aware

The implementation is a full implementation of inheritance with LSP *mostly-preserved*

Protocols / Interfaces

With PEP 544 (Py 3.8) we have structural static typing or protocol. It's like an implicitly implemented interface

How to use types

- Python >= 3.6 (3.8 for Protocols)
- Mypy

- Typed python is not a all-or-nothing feature, you can incrementally add typings as you go.
- Even without mypy or pytest-mypy, type hints will greatly improve your IDE's code completion
- You can run typed code just as regular python without any modifications or build processes.

Running MyPy

`mypy <module_name>`

- Include mypy as part of your build step CI
- MyPy will parse and print out errors if any.

Unlike Typescript, Python doesn't require preprocessing of files to remove type annotations, they are fully recognized by the interpreter and also available in the run-time with get_type_hints()

What Python Type System is not

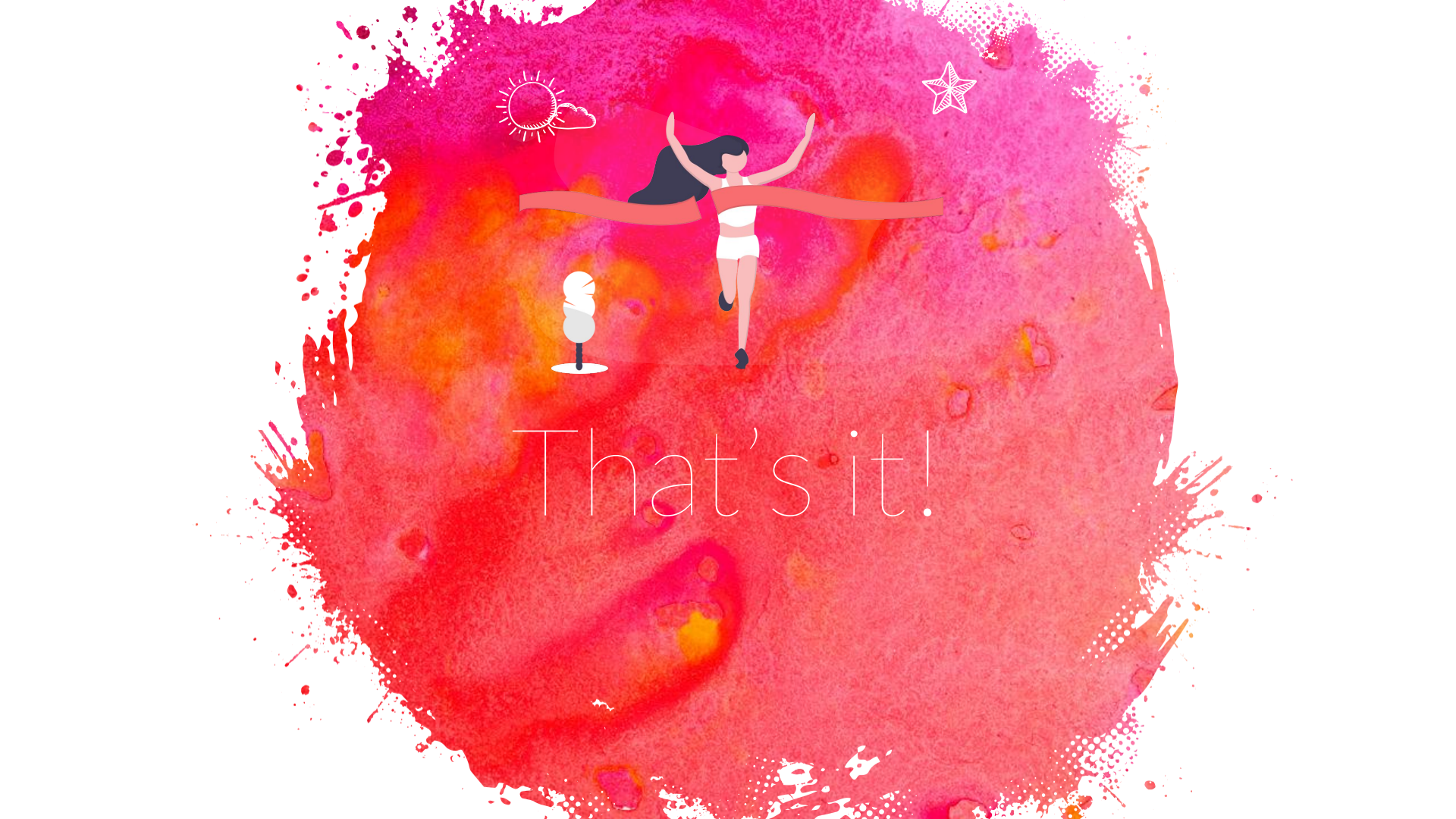
1. No run-time type checking
2. No impact on performance
3. Doesn't need any extra compilation step
4. Is not a new language like Typescript, type-annotated python is valid Python which is accepted by Python interpreters
5. Doesn't force you to use types everywhere

Some interesting usage of Python Types

1. Pydantic is a validator that is fully described by Python types, annotations and decorators
2. FastAPI is a web framework that describes it's contract through types
3. Mypyc compiles type-annotated python to cython-based native code

Beyond types

- Slots help in enforcing some of the static typeness in runtime and greatly helps in memory usage
- Retype is a tool for automatically typing a large python codebase
- Cyton creates compiled extensions which are very fast from a language very similar to python


That's it!

Any questions?

You can find me at:

@agathver

asm@amitosh.com